GPDS - A NEW SIMULATION LANGUAGE

Philip S. Becker, Jr.
Scientific/Systems Products Section
Program Products & Services Department
Xerox Data Systems

ABSTRACT

XDS has developed the General Purpose Discrete Simulator (GPDS) System which runs on its Sigma computing systems in either a time-shared or batch environment. GPDS is an extension of GPSS. Enhancements include a matrix library feature and the capability of automatically segmenting a model into overlay sections and storing these sections on a direct-access device. Other improvements to GPSS include the ability to use indirect addressing with any entity instead of just parameters, two new blocks that can locate other transactions and access data from them, and the ability to terminate a simulation when a steady-state condition is reached. This paper briefly discusses the design philosophy behind GPDS and then examines the enhancements incorporated in the language. The discussion includes some examples that illustrate the use of these new features and suggests some new modeling techniques which are now available to the user.

DESIGN PHILOSOPHY BEHIND GPDS

In designing a discrete simulation language, the following criteria were considered:

1) GPDS must be easy for nonprogrammers to use. The language should be directed to the ultimate user - the person who is to do the simulation - not toward a programmer who must interpret the modeler's intentions.

2) The language should be tightly structured. The language structure should define a solid framework for simulations and necessitate a minimum knowledge of advanced programming techniques. This was deemed more important than a more flexible language which would be more difficult to use. The language should also automatically perform such basic functions as keeping track of simulated time, sequencing of conflicting events, and recording statistics about the run.

3) The language should be compatible with existing languages. The language should minimize the conversion problems often associated with installing a new system. Also, the cost to XDS of designing, teaching, and selling a compatible language should be considerably below the cost of starting from scratch.

The language that we felt best fit these criteria was GPSS. It has been designed to be consistent with the analytic approach taken by the people who will be using the program to solve problems with simulation analysis. GPSS, of course, is a tightly structured language and it is the most widely used and best known of the existing discrete simulation languages. Almost every producer of computer mainframes has implemented some version of the language. For the best possible market penetration, it was decided to make GPDS completely compatible with GPSS/360. Any job that would run on GPSS/360 would run with no change under GPDS and the cost of the GPDS run would be significantly less.

IMPROVEMENTS TO THE LANGUAGE

As has been already mentioned in this paper, GPDS is fully compatible with the existing industry standard. However, it is the new features incorporated in GPDS that make it a valuable and unique tool. The rest of this paper details these new features and how they can be used.

Probably the most significant feature designed into GPDS is its ability to run under the Batch Time-Sharing Monitor (BTM) and under the Universal Time-Sharing System (UTS), the two XDS time-sharing operating systems. A GPDS user can build, debug, and run his model directly from any standard remote terminal, thereby eliminating the delays in turnaround that are typical of a normal batch-oriented operation.

Discrete simulation models can run for a long time with no indication of whether the model is looping or proceeding properly toward completion. During the execution phase of GPDS the remote user can check on the progress of his simulation by simply pressing the Interrupt button on this terminal. GPDS will respond by printing the simulated clock time and the number of terminations to go, and then continue the simulation. If either the clock or the terminations counter are not changing at approximately the rate expected by the user, he can terminate the simulation and check for logical loops or similar errors in the code. This allows him to abort a simulation without wasting machine time if the model "hangs up" unexpectedly at some time during the run. This interrogation feature may also be invoked from the operator's console to check the status of jobs run in a batch environment.

Since listing the output of a large run at a terminal can be a time-consuming process, if the user does not want to wait while all of his output is printed, he may assign most of his output to a disk file and direct only the output from the report generator to the terminal. He may then use the editing option of the time-sharing monitor to examine portions of the large listing file on the disk. If necessary, he may also use the terminal to cause that output file to be listed off-line on a high-speed printer.

The ability to run GPDS under the time-sharing systems has already greatly reduced the time and cost required to develop working models at XDS. We have also discovered that teaching simulation with GPDS is more effective when a student has direct access to the system via a remote terminal. With the "instant turnaround" afforded by a terminal, a student can try a new technique, find out if it works, and then modify it if necessary, all in the space of a few minutes instead of a few days.

Possibly the most severe limitation on the use of previous GPDS-type simulators was the large amount of core required for their operation. Running time - the other problem of large simulations - may inhibit repetitive runs of a large model but it is not the absolute restriction that size is. A long-running model can be used when sufficient machine time becomes available, but if a model is too big to fit in a computer, it must somehow be modified or it simply can never be used. Moderate-sized hardware is required for any significant models and it is not unusual to have to code models carefully to fit a large model into even 100,000 word machines. GPDS has incorporated features that help minimize this problem.

We found that normally in large models the majority of core required for simulation was devoted to blocks, parameters, and matrix savevalues. Therefore, GPDS allows the user to segment these entity-types into individual load sections that will be stored on a fixed-head, direct-access device known as a RAD. GPDS rolls individual load sections into core only as they are needed. New reallocation mnemonics for each entity allow the user to specify the number of that entity that will remain permanently in core and the number of each entity that comprises a single load section. Thereafter, the process is invisible to the user since GPDS automatically swaps the load sections as needed. For example, a user could allocate the first one hundred blocks as permanently core-resident and segment the rest of his model into load sections of fifty blocks each. No matter how large the model actually is, it will only require core for one hundred and fifty blocks. By properly coding his model so that the most frequently used blocks are the ones that are permanently in core and so that transaction movement between blocks in different load sections is minimized, a user could put a very large model into core space required for only one hundred and fifty blocks, yet pay a minimum in I/O overhead. To conserve more core, the user could also elect to store parameters for all transactions in load sections. To minimize I/O time, the user can also specify that a few of the more frequently used parameters in each transaction remain permanently resident in core. For a given transaction, its parameter load section would only be loaded when the model references a nonresident parameter. Likewise, the user can store both halfword and fullword matrices on separate load sections. In both cases, reallocation mnemonics are used to select which matrices are permanently core-resident and in what combination the others are stored on a RAD.

All four load section features - blocks, parameters, and the two Matrix Savevalue types - are completely independent. Therefore they may all be used without interference in one model to minimize core requirements and permit the use of large models on moderate-sized hardware.

GPDS also incorporates a library feature for Matrix Savevalues. This means that the user can produce a generalized model and define all system characters in terms of the matrices. He then builds up a library of data matrices for various systems, either directly from previous GPDS runs, or as the output of user-written FORTRAN or COBOL programs. The user then calls in the proper matrix at run time by issuing a GET directive, thereby allowing him to configure his general model to a specific situation. This feature can also be used to produce snapshots during a run by dumping data to the matrix library and analyzing it with a post-processor program.

We are already using this feature at XDS. In simulations of advanced hardware systems, we defined the various components - peripherals, processors, memory banks, and software mixes - in terms of matrices. We could then configure a specific system by calling in the proper combinations of matrices.

We are also exploring the use of the library feature in another model. This would involve defining a generalized PERT processor in terms of a GPDS model. Transactions would be generated at the start of the network, and the nodes would be simulated with SPLIT and ASSEMBLE blocks. Activity times would be ADVANCE blocks. However, the delay times would be stated as distributions which would be defined by a matrix. Enough transactions would be sent through the network to generate a series of probabilities of start and completion times for each activity. The output would be to a second matrix that would be saved for analysis and listing by a report processor to enter the data for the network and dump the results for further processing.

Many discrete simulations are primarily designed to learn the behavior during the dynamic portion of a run. When a system starts to smooth out and approach a steady-state condition, the important part of the run is complete and processing can be terminated. GPDS has introduced the SSTATE card to accomplish this. The SSTATE card allows the user to select any Standard Numerical Attribute and specify a high and low limit for this entity. When the SNA has stayed between these two values for a time period which has also been defined with the SSTATE card, processing will be terminated just as if the termination count has been decremented to zero. For example, using this command, a modeler could halt a simulation when a queue length remains within a specified range or when a utilization stays within selected limits for some desired time span.

At XDS, we have also used this technique in a simulation

of a computing system to interrupt the run when the CPU finally became overloaded. We defined a Boolean variable as CPU utilization greater than 97% and message queues greater than some constants which were varied from run to run. The arrival rate for CPU requests was gradually increased while the SSTATE card monitored the variable. When the variable remained "true" for several machine cycles, the CPU was considered overloaded and the SSTATE card halted processing. The current rate for CPU requests at the time processing was halted, therefore, becomes the maximum capacity for the CPU.

Transaction-based languages such as GPDS are excellent tools for simulating problems in which transactions act upon equipment. Event-oriented problems where something happens only when some other event has occurred can also be readily simulated by a GENERATE-GATE block combination. For example, trying to model a store where a reorder is issued only when a warehouse becomes empty could be simulated by

```
GENERATE    1
GATE  SE    WHSE
```

This code would cause a new transaction (order) to be generated whenever the storage WHSE became empty. However, transactions can test and react to the status of another transaction only in special cases. With most languages, if transactions are to interact, they must all be members of the same Group or Assembly Set.

GPDS incorporates some new features that readily enable one transaction to "communicate" with another transaction. The active transaction has the ability to access the parameters or learn the next block address of any other transaction in the system. By testing the position or parameter contents of another transaction, the modeler may now directly control the behavior of the active transaction. It is therefore possible for the active transaction in GPDS to interact directly with other transactions.

To use the transaction-communication feature, the user must reference a specific transaction by its transaction number. The number of the active transaction may be found by referencing the SNA "XN1". Typically, a transaction will store its transaction number in a savevalue. Thereafter, any other transaction may check the status of the first transaction by using the contents of the savevalue to identify the first transaction.

The LOCATE block permits an active transaction to store the next block address of any other transaction, in either a savevalue or a parameter of the active transaction. The USING block allows the active transaction to copy the contents of any parameter of any other transaction into a savevalue or into a parameter of the active transaction. Also for both blocks, the user can select an alternate address for the entering transaction if the selected transaction is not currently used in the model.

These two blocks allow a variety of new techniques. For example, assume that in a message-switching simulation with signals arriving from a variety of sources, a signal would be delayed if the one immediately preceding it has not stayed far enough ahead at certain key locations. The LOCATE blocks can be used to check the location of the preceding transactions in the following manner:

```
        ASSIGN      2,X1        Store transaction
                                number of preced-
                                ing transaction in
                                parameter two.

        SAVEVALUE   1,XN1       Save number of
                                the transaction
                                to pick up.

        LOCATE      P2,P,5      Store the next
                                block address of
                                the preceding
                                transaction in P5.

        VARIABLE    P5-AAA      Calculate number
                                of blocks the pre-
                                ceding transaction
                                is ahead.

AAA     TEST L      V1,7,BBB    See if preceding
                                transaction is less
                                than 7 blocks
                                ahead.

        ADVANCE     25          If not 7 blocks
                                ahead, wait 25
                                time units.

BBB     ---------               Continue process-
                                ing.
```

In a similar fashion, the USING block can be used to control the flow of a transaction, based on the contents of a parameter from another transaction. All transactions can reserve a parameter for use as a progress flag. As the transaction completes some simulated process, it would update the parameter to indicate its new status. Thereafter, any transaction can determine the status of another transaction by examining that parameter.

Thus with the LOCATE and USING blocks, the number can now readily base the control logic for one transaction on the status - either the location or contents of a parameter - of another transaction.

GPSS-type languages generally permit the user to refer to entity numbers indirectly addressed by the contents of some specified parameter. GPDS has expanded this technique to allow the modeler to use any entity for indirect specification. The operator "#" is used to indicate this expanded indirect specification much as the "*" is used to mean indirect based on the value of a parameter. However, the entity type to be used as the

base is not implicit in the "#". The default is a fullword savevalue. Thus, if the entity base has not been redefined,

```
SEIZE    #4
```

would mean "Seize the facility whose number is given in fullword savevalue four".

The SET directive is used to redefine the implied entity type (entity base) for the new indirect operator. When the Assembly phase encounters a SET directive, it replaces the current entity base with the entity type indicated by the Standard Numerical Attribute given in the argument field. This new entity-base remains in effect until the Assembly phase encounters the next SET directive.

Thus, the code

```
SET      V
SEIZE    X#2
DEPART   X#1
SET      FN
ADVANCE  XH#3
```

would be treated as "Seize the facility whose number is given in the savevalue indicated by variable two, depart from the queue whose number is given by the savevalue whose number is the value of variable number one, then delay the number of time units given in the halfword savevalue pointed to by function number three".

Expanded indirect addressing greatly simplifies the development of GPDS models. The previous procedure to create the effect of indirect addressing with entities other than a parameter would be first to use an ASSIGN block in order to store the value of the other entity in a parameter and then to address from that parameter. Such a procedure would add just one more small but unnecessary complication to a simulation. This technique also usually required that the programmer reserve an extra parameter for this step in every transaction that might require that type of indirect addressing. In addition, it meant that each time that code was executed, the transaction would have to pass through an extra block. By eliminating the need for the extra parameter for every transaction and the extra block for each case of indirect addressing, expanded indirect addressing can yield a substantial decrease in both the core requirement and the processing time for a moderate-sized model.

GPDS has some other new features. The modeler may now use the REALLOCATE mnemonic "RND" to allocate up to ninety-nine random generators. This feature is important in large models where random distributions are used in many blocks and each use must be independent of all others. A large number of generators must be available so that the model may be rerun with different orders of calls to the generators without interfering with the sequence of numbers returned from a single generator.

Since random number sequences are such an important part of GPDS simulation, considerable effort went into the development of an effective random number generator. The algorithm that is used in GPDS is described in The Art of Computer Programming, Volume 2[1] by Donald C. Knuth. It is of the form $R2 = R1\ A + B$ where R1 is the previous number or the seed and A and B are constants that were selected based on criteria specified in Knuth's book.

This algorithm will produce $2^{31}-1$ numbers before repeating. A variety of checks were made to assume random sequences. These include the equidistribution test to assure a uniform distribution, serial tests of pairs, triples, and quadruples to assure that subsequent sets of numbers are independent, and a version of the chi-square test to verify randomness of the sequences. Standard correlation tests were also used to check that the correlation between a number and any of the next three was low.

The normal and negative exponential distributions are now built into GPDS in the form of two twenty-eight-point functions. They are defined by standard FUNCTION cards, but require no follower cards. They are indicated by the mnemonics "BN" and "BE" respectively in the B-field. The normal curve has a mean of zero and standard deviation of one. The negative exponential has both a mean and a standard deviation of ten. The formula used to calculate the exponential curve is

$$f(x) = 0.1e^{-0.1x}$$

where x is the value of each point. The functions are usually referenced in a variable statement to transform its characteristics. For example, to create a normal distribution with a mean of seventy-five and a standard deviation of sixty would require:

```
NORM VARIABLE   60*FN1 + 75
1        FUNCTION RN1, BN
```

Thereafter, references to V$NORM would produce sample values from a normally distributed random set of numbers with a mean of seventy-five and a standard deviation of sixty.

The GPDS Variable Statement itself has been expanded. The exponential operator ":" and the square root operator " | " have been added to aid in defining involved mathematical relationships. These two operators are processed

References

(1) Donald E. Knuth, The Art of Computer Programming, Volume 2/Seminumerical Algorithms (Menlo Park: Addison Wesley, 1969), Chapter 3.

in the same order as is the exponential operator in FORTRAN. That is, they have identical priorities that are higher than those of all other operators. Among other benefits, the new operators allow the user to express non-linear relationships in terms of exact mathematical statements instead of approximating them with FUNCTION statements.

Also, all variable statements are now calculated using Reverse Polish Notation. The Assembly phase of GPDS produces strings of parenthesis-free data using this notation. The stack hardware on the XDS Sigma equipment is used to store the strings during assembly and later to retrieve them during the Execution phase. This results in faster calculations and smaller core requirements than with previous techniques. It also eliminates the restriction on the number of pairs of parentheses allowed in a variable statement.

GPDS has three HELP blocks to simplify the interface with FORTRAN, COBOL,and Meta-Symbol, the XDS assembly language. HELPF and HELPC allow direct calls to FORTRAN and COBOL respectively. The A-field of each HELPF or HELPC block contains the name of the subroutine and the B through G fields contain the data to be passed to the subroutine. Any Standard Numerical Attribute can be passed to a subroutine and the subroutine can directly alter savevalues, matrix savevalues,and parameters. The standard report writer DCB is directly callable by any HELP block,so all special reports can be directed to a single output device or file if the user so desires.

The basic HELP block is used to transfer control to Meta-Symbol routines. The user may share data with a Meta-Symbol routine in two ways. He may use the B through G fields of the HELP block,letting GPDS set up an argument list for him to simplify passing data to the subroutine. To accommodate more complex interactions between GPDS and a subroutine, the pointers to all GPDS entities and internal tables are kept in a single DSECT. A Meta-Symbol user can create a copy of the DSECT in his own program simply by using the command SYSTEM CNTRLWRDS, Meta-Symbol will then automatically build a copy of the DSECT which the programmer then references to learn the address of any GPDS entity. Thus a single command gives the user direct access to all of GPDS.

SUMMARY

This then, is an overview of the Xerox General Purpose Discrete Simulator. By starting with the most popular simulation language in use today, then adding features that have been sought by users, and improving in areas where competitive programs were weakest, XDS feels that they have developed a discrete simulation language that can become the standard in its field. GPDS, along with SL-1 and FMPS/GAMMA III, now gives the users of XDS equipment a selection of simulation software that equals any in the industry.